GAME 999 Backend

1. ✅ API structure by role (User / Admin / Agent)
2. 🛠️ Endpoint details
3. 🗃️ MongoDB schema (data models)
4. 🔒 Validation rules
5. 🔑 Auth and security
6. 📋 General notes for your backend developer

✅ 1. API STRUCTURE OVERVIEW

USER ROLE

Functionality	Endpoint	Method
Register	`/api/register`	POST
Login	`/api/login`	POST(user→mobile number/username)
Get Profile	`/api/profile`	GET
Update Profile	`/api/profile`	PUT
Dashboard	`/api/dashboard`	GET
Place Bet	`/api/game/bet`	POST
Get Game Rounds	`/api/game/rounds`	GET
Get Result	`/api/result?round={}`	GET
Bet History	`/api/history`	GET

Wallet Summary	`/api/wallet`	GET
Add Token	`/api/wallet/add`	POST
Withdraw Token	`/api/wallet/withdraw`	POST
Get Notifications	`/api/notifications`	GET

ADMIN ROLE

Functionality	Endpoint	Method
Login	`/api/admin/login`	POST
Manage Users	`/api/admin/users`	GET/POST
Update/Delete User	`/api/admin/users/{id}`	PUT/DELETE
Manage Tokens	`/api/admin/tokens/add`	POST
	`/api/admin/tokens/withdraw`	POST
	`/api/admin/tokens/approve`	PUT
Result Management	`/api/admin/results`	POST
	`/api/admin/results?round={}`	GET
Upload QR Code	`/api/admin/qr`	POST
Login Logs	`/api/admin/logs`	GET
Manage Agents	`/api/admin/agents`	GET/POST
Update/Delete Agent	`/api/admin/agents/{id}`	PUT/DELETE
History & Analytics	`/api/admin/history`	GET
	`/api/admin/analytics`	GET

AGENT ROLE

Functionality	Endpoint	Method
Login	`/api/agent/login`	POST
Agent Dashboard	`/api/agent/dashboard`	GET
Add User	`/api/agent/add-user`	POST
View Users	`/api/agent/users`	GET
View Bets	`/api/agent/bets?userId={}`	GET
View Transactions	`/api/agent/transactions?userId={}`	GET
View Results	`/api/agent/results?userId={}`	GET

🗃️ 2. MONGODB SCHEMAS

📄 User Schema

```
{
  _id,
  fullName: String,
  mobile: { type: String, unique: true },
  email: String,
  password: String,
  balance: Number,
  referralCode: String,
  agentId: ObjectId,
  createdAt: Date
}
```

📄 Agent Schema

```
{
  _id,
  fullName: String,
  mobile: { type: String, unique: true },
  password: String,
  referralCode: String,
  users: [ObjectId],
  createdAt: Date
```

```
}
```

📄 Admin Schema

```
{
  _id,
  fullName: String,
  mobile: { type: String, unique: true },
  password: String,
  role: String,
  createdAt: Date
}
```

📄 Bet Schema

```
{
  _id,
  userId: ObjectId,
  roundNumber: Number,
  numbers: [Number],
  amount: Number,
  timeSlot: String,
  status: String, // placed, won, lost
  createdAt: Date
}
```

📄 Round Schema

```
{
  roundNumber: Number,
  startTime: Date,
  endTime: Date,
  resultNumber: Number,
  status: String // pending, completed
}
```

📄 Transaction Schema

```
{
  _id,
  userId: ObjectId,
  type: String, // add, withdraw, bet, win, loss
  amount: Number,
  status: String, // pending, approved, rejected
  createdAt: Date
}
```

📄 Notification Schema

```
{
  _id,
  title: String,
  message: String,
  type: String, // info, warning, success
  createdAt: Date
}
```

📄 QR Code Schema

```
{
  _id,
  image: String,
  createdAt: Date,
  updatedAt: Date
}
```

📄 Audit Log Schema

```
{
  _id,
  userId: ObjectId,
  adminId: ObjectId,
  action: String,
  timestamp: Date,
  details: String
}
```

🔒 3. VALIDATION RULES

- `mobile`: must be 10 digits, unique
- `password`: minimum 6 characters, hashed with bcrypt
- `numbers[]`: only between 1-99, max 4 per round
- `amount`: must be within defined min/max (set in config)
- `email`: must be valid if provided

- `referralCode`: must exist in agent collection if provided
- `profilePhoto`: must be image file
- `round timings`: system logic enforces time slots
- `agent`: only sees users linked to them
- `admin`: sees all data

🔑 4. AUTH & SECURITY

- 🔐 **JWT tokens**:
  - User: `Bearer <user_token>`
  - Admin: `Bearer <admin_token>`
  - Agent: `Bearer <agent_token>`
- 🧮 Role-based middleware to verify access to routes:
  - `/api/*` = user
  - `/api/admin/*` = admin only
  - `/api/agent/*` = agent only
- 🔐 Password hashing using **bcrypt**
- 📜 Input validation using **Joi** / **express-validator**
- ✅ Secure file uploads using **Multer**

⏱️ 5. GAME TIMING LOGIC

- Game runs **11 AM to 12 AM**

- Each hour = 1 round
    - **First 50 minutes**: bet placement
    - **Last 10 minutes**:
        - Admin can declare result
        - If no action in 5 mins → auto-generate using algorithm
- Results visible after each round ends
- Backend cron or queue service (e.g. node-cron) should handle this

---

📋 6. GENERAL NOTES

- 📦 All endpoints return JSON responses with `{ success, message, data }`
- ❌ Clear error messages on failure
- 📌 Audit logs for every login/logout/update/delete
- 🛠️ Admin can override auto-result
- 👩‍💻 Super Admin can see everything
- 👥 Agents are only allowed to manage their users
- ⏱️ All server time handling should be **timezone-consistent (e.g., IST)**

⚙️ OPTIONAL: Additional Configs

- 📧 Email verification (via nodemailer)
- 🔐 Rate limiting for login
- 🪙 Wallet balance locking during bet
- 🔁 Retry mechanism for payment gateways

- 📤 Export to CSV for admin analytics
- 📲 Push notifications (via Firebase or OneSignal)

🎯 GOAL:

Minimize the number of winners per round by:

- **Only allowing result selection** from the **4 least-picked numbers** by users in that round.
- If the admin does not pick manually in 5 minutes, system **auto-generates** from the same 4 numbers using a strategy.

📘 RULE OVERVIEW

1. Each round allows bets on numbers (1-99).
2. Each user selects up to 4 numbers.
3. At the end of a round, calculate **frequency** of all selected numbers.
4. Pick the **4 numbers with the lowest frequency**.
5. Admin can choose **only one** from these 4.
6. If admin doesn't select in 5 minutes → system selects randomly or based on a defined algorithm.
7. This ensures **maximum profit** for admin (since least people bet on these numbers).

⚙️ ALGORITHM

🔹 STEP 1: Collect All Bets for Round

```
// Fetch all bets for current round
const bets = await Bet.find({ roundNumber });
```

🔹 STEP 2: Count Number Frequencies

```
const numberFrequency = Array(100).fill(0); // index 1 to 99

bets.forEach(bet => {
  bet.numbers.forEach(num => {
    numberFrequency[num]++;
  });
});
```

🔹 STEP 3: Find 4 Least Picked Numbers

```
const numberFreqArray = numberFrequency
  .map((count, number) => ({ number, count }))
  .filter(n => n.number >= 1 && n.number <= 99);

// Sort by least picked
numberFreqArray.sort((a, b) => a.count - b.count);

// Get 4 least picked numbers
const leastPicked = numberFreqArray.slice(0, 4).map(n => n.number);
```

🔹 STEP 4: Admin Manual Selection

In admin panel:

- Show only the 4 numbers in dropdown/radio UI
- Allow only one result selection
- Save selected number to Round schema: `resultNumber`

🔹 STEP 5: Auto Selection if Admin Doesn't Pick

Time logic:

- Result selection starts at minute 50 of every hour
- If no result set by minute 55, auto-generate

```
if (!round.resultNumber && Date.now() > round.startTime + 55 minutes) {
  const selected = autoPick(leastPicked);
  round.resultNumber = selected;
  round.status = 'completed';
  await round.save();
}
```

🔹 Auto Pick Logic

You can apply one of these:

1. Pure Random

```
function autoPick(numbers) {
  const randomIndex = Math.floor(Math.random() * numbers.length);
  return numbers[randomIndex];
}
```

2. Pick the Number with Zero Selections

```
function autoPick(numbers, numberFrequency) {
  return numbers.find(num => numberFrequency[num] === 0) || numbers[0];
}
```

3. Fixed Strategy (Always choose min number)

```
function autoPick(numbers) {
  return Math.min(...numbers);
}
```

🛡️ SECURITY / FAIRNESS NOTES

- **Do not allow admin to override least-picked list** (read-only)
- Log who selected the result (admin ID or system)
- If auto-generated → mark `generatedBy: "system"` in audit log

- Users can see declared result but not the least-picked list (prevent reverse engineering)

📊 Example

Bets:

- User A: [5, 17, 25, 36]
- User B: [5, 8, 12, 25]
- User C: [5, 17, 50, 60]
- User D: [8, 9, 10, 12]

Frequency Count (1-99):

5: 3
8: 2
9: 1
10: 1
12: 2
17: 2
25: 2
36: 1
50: 1
60: 1
// rest: 0

Least Picked Numbers:
→ From above, least picked = [9, 10, 36, 50] (all picked only once)

Admin sees these 4 numbers.
If admin doesn't pick → system picks randomly from those.
